Number Theory Behind and Against RSA Encryption

The August 1977 issue of the Scientific American magazine included a publication by Ron ***R***ivest, Adi ***S***hamir, and Len ***A***dleman about a new cryptosystem which would go on to be such an integral advancement in digital security: ***RSA*** encryption. It still plays a vital role today, keeping transferred data secure when you visit websites, in email services and much more. Even better than this 'security' stuff, it grants us the opportunity to take a look at some beautiful number theory. We can then delve into some satisfying mathematical ways to try to crack RSA encryption.

You may presume the described cryptosystem relies on some highly esoteric, cutting-edge research on mathematics and computer science. However, it turns out RSA encryption is simply built on the fact that reversing the multiplication of two large prime numbers is computationally infeasible. This may not seem obvious, so here is an example:

$$71 \times 97 = 6887$$

$$p_1 \times p_2 = 5251$$

The first equation shows the simplicity of multiplying two primes. But how can we reverse the process to find which two primes multiply to give 5,251? Unfortunately (or fortunately for cryptographers) this is not quite as easy. Now consider factorising the product of two 100-digit primes. With a non-quantum computer and a brute-force method, you would be there for longer than the age of the universe.

So how do we take this fact and turn it into a convenient and efficient way of encrypting and decrypting data? That's what the RSA algorithm solved. Before getting into its details, let's first look at some of Euler's work.

Euler's Totient Function

Let's start by defining Euler's totient function, φ(**n**) ('phi of **n**'). It counts the number of positive integers up to n that are relatively prime to **n**. For a number to be relatively prime to **n**, it must not share any factors with **n** other than one. It is easiest to understand this with a couple of examples:

$$\varphi(9) = 6$$

As 1, 2, 4, 5, 7 and 8 are all relatively prime to 9.

$$\varphi(7) = 6$$

$$\text{As } 1, 2, 3, 4, 5 \text{ and } 6 \text{ are all relatively prime to } 7.$$

Notice that the totient of any prime number, **p**, will always be (**p-1)** because it has no factors other than 1 and itself, so all numbers less than **p** *must* be relatively prime to **p**.

Euler's Totient Theorem

In 1640, Pierre de Fermat stated his 'Little Theorem' in a letter to a friend. A proof came from Leonhard Euler 96 years later (which was thankfully a little more straightforward than the proof of Fermat's *Last* Theorem), followed by his generalisation of it into Euler's totient theorem:

$$gcd(a, n) = 1 \iff a^{\varphi(n)} \equiv 1 \pmod{n}$$

In other words, if (and only if) two numbers, **a** and **n**, are coprime (their only shared factor is 1), **a** raised to the totient of **n** will give a result congruent to 1 in modulo **n**. For example,

$$7^{\varphi(3)} \equiv 1 \pmod{3}$$

$$7^2 \equiv 1 \pmod{3}$$

The RSA Encryption Method [1] [3]

Here is a brief step-by-step rundown of how we can encrypt data with the RSA method:

1. Choose **p** and **q** as two large prime numbers.

   The strength of the encryption is determined by the size and predictability of p and q. In practice, they are selected as randomly as they can be, each around 100 digits.

2. Multiply them together: **n = p × q**.

3. Calculate Euler's totient function of n. We can shortcut this using the fact the totient function is multiplicative:

   $$\varphi(mn) = \varphi(m)\varphi(n) \quad \text{if } \gcd(m, n) = 1$$

   Therefore, the totient of n is equal to the totient of p multiplied by the totient of q. Recall that the totient of a prime number is p-1, hence:

   $$\varphi(n) = (p - 1)(q - 1)$$

4. Select a public exponent (**e**) as an integer such that:

$$1 < e < \varphi(n) \quad and \quad gcd(e, \varphi(n)) = 1$$

   In practice, **e** is typically 65537.

5. Calculate **d**, the decryption key, as the modular inverse of **e** modulo φ(**n**). We can do this by finding the value of **d** such that when multiplied by **e**, it is congruent to one modulo φ(**n**).

$$ed \equiv 1 \ (mod \ \varphi(n))$$

   For small numbers this is trivial:

$$3d \equiv 1 \ (mod 40)$$

$$d = 27$$

   To find **d** when the numbers are larger, we use the Extended Euclidean Algorithm. I won't get into this here, but there are plenty of resources available [2].

We now have everything we need. If we publish **n** and **e** as the public keys, so they are freely available to see and use, anyone who wants to send us a message, **M**, can use the following formula to turn it into cipher, **C**.

$$C \equiv M^e \ (mod \ n)$$

They can then send us their ciphered data. Once we receive it, we can decrypt it back to the original message using:

$$M \equiv C^d \ (mod \ n)$$

Notice that to decrypt the message, **d** is needed, which can only be found by someone who knows the totient of **n**, which can only be found by someone who knows **p** and **q**. This is the fundamental idea behind the encryption: if anyone other than the intended recipient gets hold of the cipher text, they will not be able to translate it unless they manage to factorise **n** (and therefore will not be able to steal any information). In other words, as long as the factorisation is kept private, the message is safe (in theory).

The maths behind this works as follows:

We previously chose **e** and **d** such that

$$ed \equiv 1 \ (mod\ \varphi(n))$$

We can rewrite this as:

$$ed = 1 + k\ \varphi(n)$$

$We\ encrypt\ the\ message, M, as\ the\ remainder\ after\ raising\ it\ to\ the\ power\ of\ e$:

$$C \equiv M^e\ (mod\ n)$$

$If\ we\ decrypt\ C\ using\ our\ formula$:

M′ denoting here the result of our decryption (which should hopefully turn out to be the same as ***M***)

$$M' \equiv C^d\ (mod\ n)$$

Substituting C with M^e

$$M' \equiv M^{ed}\ (mod\ n)$$

Substituting *ed* with the equivalence shown above

$$M' \equiv M^{1+k\varphi(n)} (mod\ n)$$

$$M' \equiv M \cdot M^{k\varphi(n)} (mod\ n)$$

$If\ gcd(m,n) = 1$, * ($Euler's\ totient\ theorem$ applies)

$$M^{\varphi(n)} \equiv 1 \ (mod\ n)$$

Applying Euler's totient theorem:

$$M' \equiv M \cdot 1^k\ (mod\ n)$$

$$M' = M$$

We can loose the (mod n) since M < n

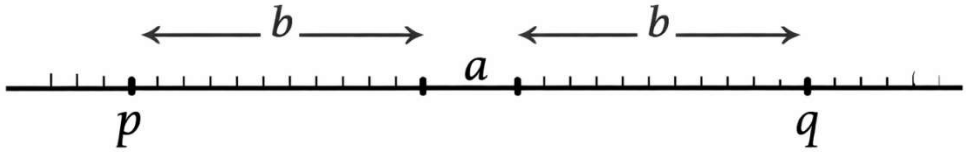
***** Note: this reasoning applies when **m** and **n** are coprime (since this is required to apply Euler's totient theorem). A slightly more complicated (but similar) method can be applied when they are not coprime.

We have hereby shown that our decryption formula yields a result identical to the original message. In this way, the RSA encryption method elegantly incorporates several elements of number theory to construct a mechanism for reliably encrypting and decrypting data.

While RSA is certainly a highly secure encryption method, it is only inevitable that people have found ways to crack its encryption. Most of the methods exploit poor selections of **p** and **q** so that a hacker is able to find the factorisation of **n** without using a brute force method. As a disclaimer, these ways of 'beating' RSA should not make you concerned about the safety of your data – it is just fun to use some clever number theory to see how we *could* undermine a widely used encryption method.

Fermat's Factorisation Method [3] [5]

Firstly, let's take a look back at some work from the great Pierre de Fermat and how we can apply it in the context of RSA encryption. He developed a factorisation method that works astoundingly efficiently on a composite number made up of two primes that are near to each other. In other words, for this method to work, **p** and **q** must have a small difference between them. Fermat starts by defining **a** as the integer directly in the middle of **p** and **q**, and **b** as the difference between **a** and each prime.



We can now write **p** as (**a-b**) and **q** as (**a+b**), hence **n** can be rewritten as a difference of two squares:

$$N = p \cdot q$$

$$N = (a - b)(a + b)$$

$$N = a^2 - b^2$$

$$b^2 = a^2 - N$$

If **a** and **b** are close to each other, **a** must be greater than or equal to the square root of N (and will be reasonably close to it). In the case where **a** is equal to the square root of N, **b** is zero, so **p** is equal to **q**. If this is not the case, **a** must be greater than the square root of N, so we can start guessing from there. With each guess, we calculate **b** and see if it is an integer value. If it is, we have found our values of **a** and **b**. If not, increment **a** by one and repeat the process until **b** is an integer. There is an excellent video on the Computerphile YouTube channel where Dr. Mike Pound writes just a few lines of code to almost instantly factorise a product of two nearby primes [4]. Fortunately, the chance of two randomly generated primes being close enough for this to be relevant is negligible. However, computers cannot produce *truly* random numbers, nor can we as humans. In this way, it is possible that pseudorandom ('random' numbers generated by a computer) values of **p** and **q** are very close to each other, making them vulnerable to this attack, making use of some centuries-old maths from Fermat.

Shared Primes Attack

Focusing again on a computer's inability to produce random numbers, it is a concern that flawed pseudorandom number generators (PRNGs) can cause multiple values of N to share a factor. For example, if two parties use similarly flawed PRNGs, they could both generate an identical prime:

$$N_1 = p \cdot q_1$$

$$N_2 = p \cdot q_2$$

In this case, although the values of N are different, they are both composed of an identical prime factor, **p**. This becomes a problem because we can efficiently compute the greatest common divisor of two numbers.

$$\gcd(N_1, N_2) = p$$

From here, it is trivial to calculate the other prime factor of each N value:

$$q_1 = \frac{N_1}{p}$$

$$q_2 = \frac{N_2}{p}$$

Many researchers in the past have scanned huge data sets containing values of N that are used in RSA encryptions. They run algorithms to find the greatest common divisor of each pairing of N values. Any results that come back as a value not equal to one must share a prime factor, so can easily be cracked. A paper by researchers at KeyFactor found that around 1 in 172 RSA implementations (on specific devices) could be cracked in this way [6].

Håstad Broadcast Attack [7]

The final approach to cracking RSA I would like to visit is not much of a threat anymore but has some fascinating number theory behind it. John Håstad proposed his 'Håstad broadcast attack' in 1988. It relies on a few conditions of the RSA implementation being met:

- A small value of **e**
- The same message, **m,** being sent to multiple recipients
- The size of **m** being sufficiently small

The same **m** may be sent to multiple recipients in a few real-life scenarios. Imagine many hardware devices all going through the same update: they all receive an

identical update token sent to their unique encryption keys. In this situation, if **e** is small (3, for example), we can create a list of simultaneous congruences as follows:

$$C \equiv M^e \ (mod\ N)$$

We have: $e = 3$, unique N values from 3 different devices and the corresponding ciphers, C

$$m^3 \ (\mathrm{mod}\ 6) \equiv 5$$

$$m^3 \ (\mathrm{mod}\ 35) \equiv 20$$

$$m^3 \ (\mathrm{mod}\ 143) \equiv 125$$

By using the Chinese Remainder Theorem (CRT), we can deduce that:

$$m^3 = 125$$

$$m = 5$$

I won't get into how CRT works here, but resources are plentiful online. It is essentially used to provide a solution to a system of simultaneous linear congruences with pairwise coprime moduli. Right at the beginning of RSA implementation, this method was a real threat. These days, larger values of **e** are used and messages are scrambled before the encryption (a process known as 'padding'), so the same **m** value would not be sent multiple times, meaning this cannot be used to crack any effectively implemented RSA encryptions. Nevertheless, this method still serves as an intriguing application of number theory to theoretically crack RSA encryptions. I wanted to raise this technique because it differs from most of the others in the sense that it is not centred around factorising **N** into **p** and **q**.

This brings us to the end of our journey exploring some of the number theory behind the renowned RSA encryption algorithm and a few approaches we can take to compromise it. Crucially, all these apparent weaknesses of RSA do not arise from the maths itself but from our flawed implementations of the system. Our journey therefore illustrates the elegance of mathematics and how we, as humans, have both the pleasure and challenge of utilising it effectively.

Bibliography

[1] Luo, Z.J. *et al.* (2024) *Demystifying the RSA algorithm: An intuitive introduction for novices in Cybersecurity*, *arXiv.org*. Available at: https://arxiv.org/abs/2308.02785 (Accessed: 06 April 2026).

[2] Extended Euclidean Algorithm (find on page 164) - Diko, E. and Ibraimi, M. (2023) *RSA & Extended Euclidean algorithm*, *ResearchGate*. Available at: https://www.researchgate.net/publication/370116281_RSA_EXTENDED_EUCLIDEAN_ALGORITHM_WITH_EXAMPLES_OF_EXPONENTIAL_RSA_CIPHERS_RSA_EXAMPLE_SOLUTION_WITH_EXTENDED_EUCLIDEAN_ALGORITHM (Accessed: 12 April 2026).

[3] Stearn, J. (2021) *RSA problem and factorisation algorithms*, *Scribd*. Available at: https://www.scribd.com/document/825712637/JStearn-Dissertation (Accessed: 09 April 2026).

[4] Breaking RSA - Computerphile, Dr. Mike Pound (Accessed: 10 April 2026).

[5] Fermat's Factorisation - Böck, H. (2023) *Fermat factorization in the wild*, *IACR Cryptology ePrint Archive*. Available at: https://eprint.iacr.org/2023/026 (Accessed: 10 April 2026).

[6] Shared Primes - Kilgallin, J. and Vasko, R. (2019) *Factoring RSA keys in the IOT ERA*, . Available at: https://ieeexplore.ieee.org/document/9014350 (Accessed: 11 April 2026).

[7] Cracking RSA with CRT, Bill Buchanan OBE (Accessed: 13 April 2026).